

Unidade III:

Fundamentos de Análise de Algoritmos



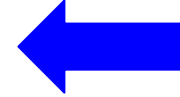
PUC Minas

Instituto de Ciências Exatas e Informática
Departamento de Ciência da Computação

Agenda

- Potência, Logaritmo, Piso e Teto, e Função
- Contagem de operações
- Aspectos da análise de algoritmos
- Função de complexidade
- Notações O , Ω e Θ

- **Potência, Logaritmo, Piso e Teto, e Função**
- Contagem de operações
- Aspectos da análise de algoritmos
- Função de complexidade
- Notações O , Ω e Θ



Exercício Resolvido (1)

- Resolva as equações abaixo:

a) $2^{10} =$

b) $\lg(1024) =$

c) $\lg(17) =$

d) $\lceil \lg(17) \rceil =$

e) $\lfloor \lg(17) \rfloor =$

Nota: $\lg(n)$ é a mesma coisa que o logaritmo de n na base dois, ou seja, $\log_2(n)$

Exercício Resolvido (1)

- Resolva as equações abaixo:

a) $2^{10} = 1024$

b) $\lg(1024) = 10$

c) $\lg(17) = 4,08746284125034$

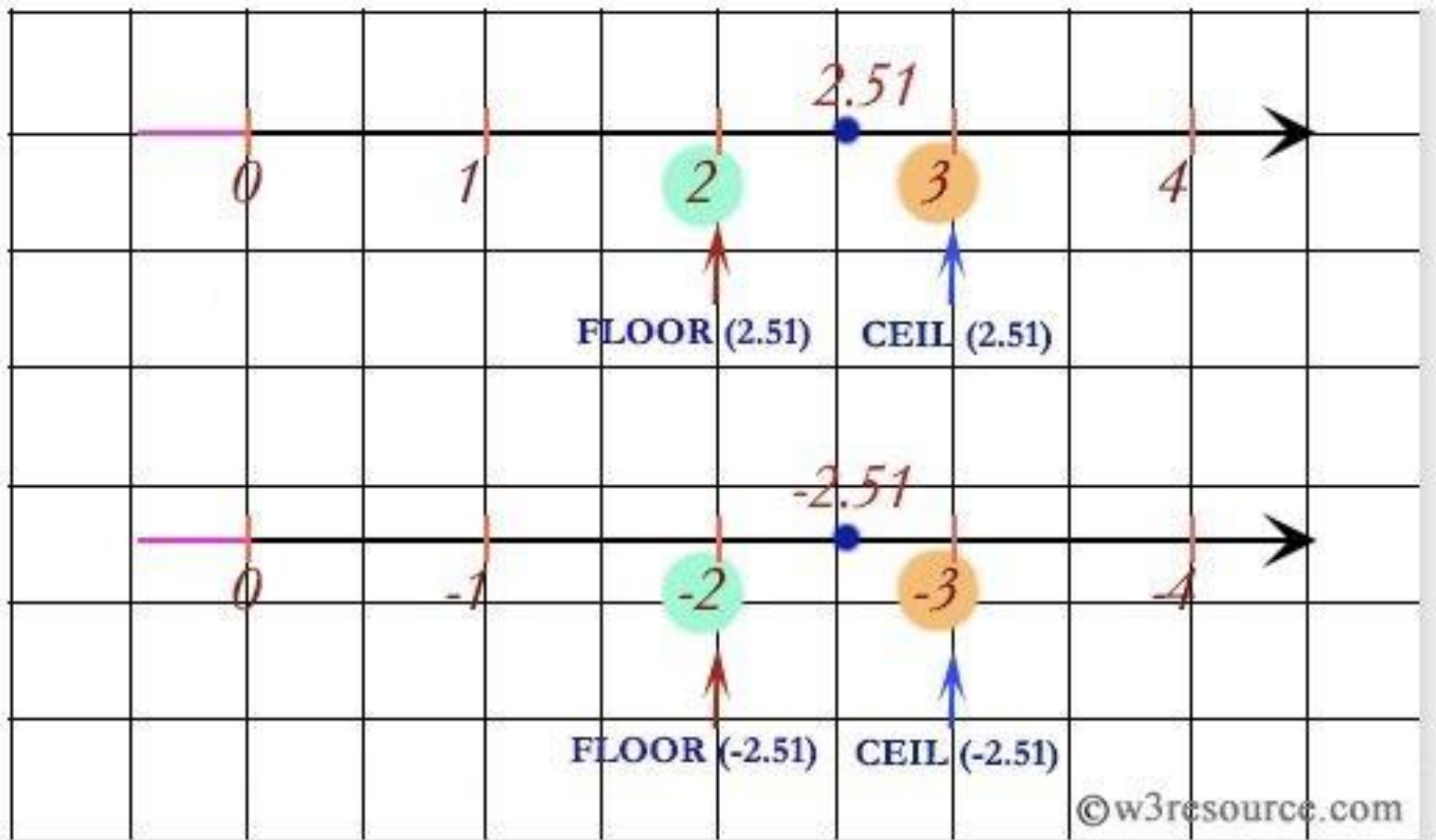
d) $\lceil \lg(17) \rceil = 5$

e) $\lfloor \lg(17) \rfloor = 4$



Nota: $\lg(n)$ é a mesma coisa que o logaritmo de n na base dois, ou seja, $\log_2(n)$

Piso e Teto



Exercício Resolvido (2)

- Plote um gráfico com todas as funções abaixo:

a) $f(n) = n^3$

b) $f(n) = n^2$

c) $f(n) = n \times \lg(n)$

d) $f(n) = n$

e) $f(n) = \text{sqrt}(n)$

f) $f(n) = \lg(n)$

Exercício Resolvido (2)

- Plote um gráfico com todas as funções abaixo:

a) $f(n) = n^3$

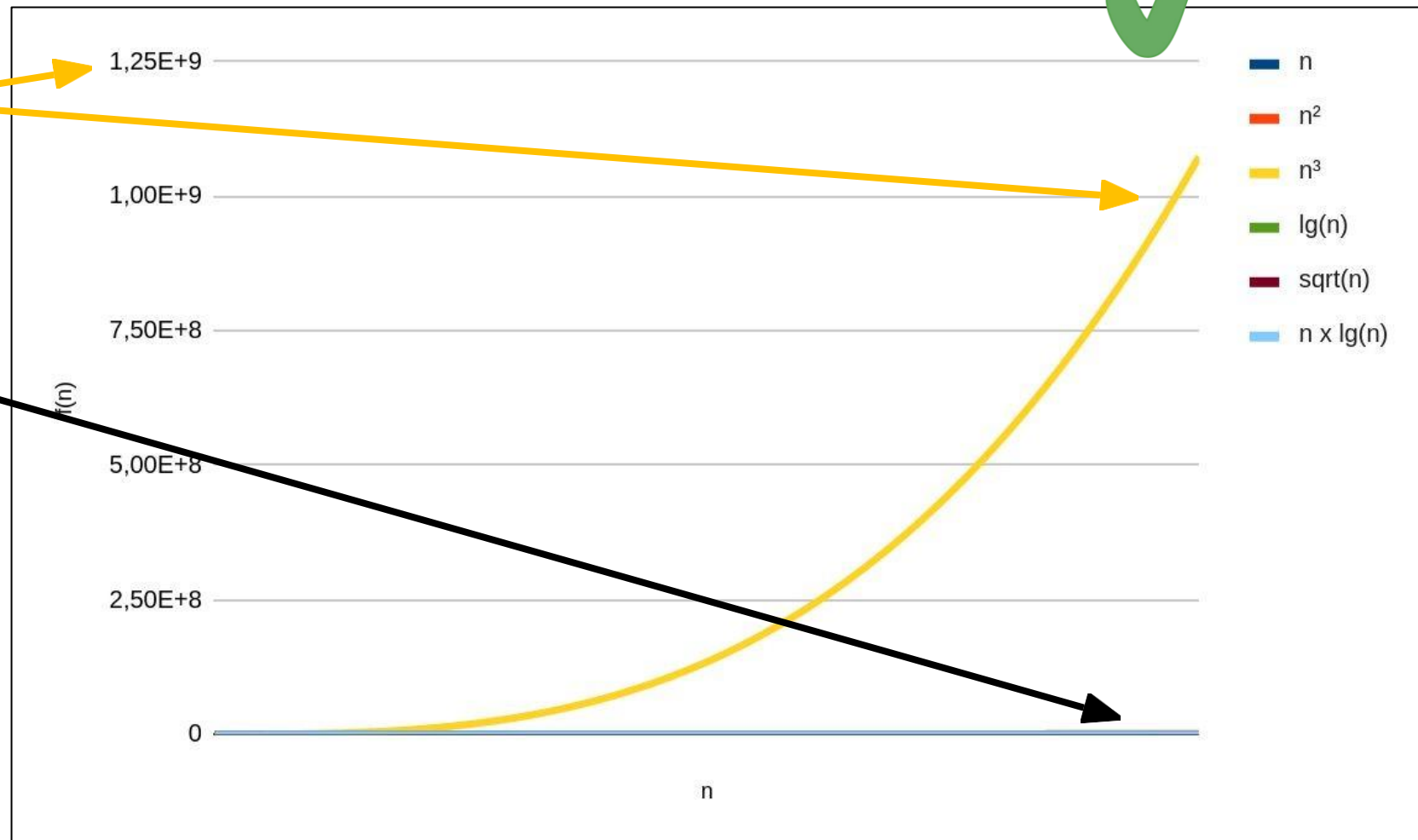
b) $f(n) = n^2$

c) $f(n) = n \times \lg(n)$

d) $f(n) = n$

e) $f(n) = \text{sqrt}(n)$

f) $f(n) = \lg(n)$



Exercício Resolvido (2)

- Plote um gráfico com todas as funções abaixo:

a) $f(n) = n^3$

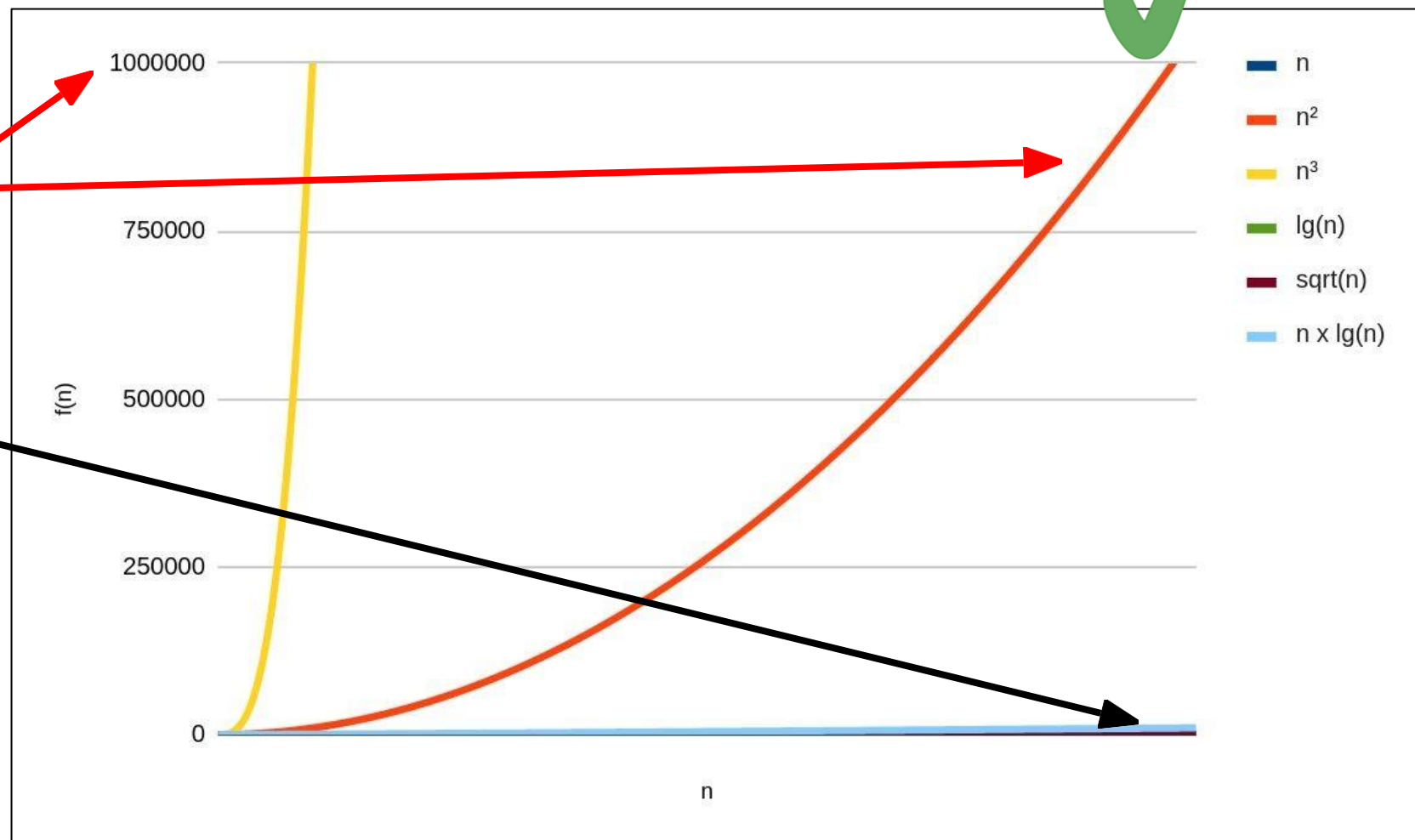
b) $f(n) = n^2$

c) $f(n) = n \times \lg(n)$

d) $f(n) = n$

e) $f(n) = \text{sqrt}(n)$

f) $f(n) = \lg(n)$



Exercício Resolvido (2)

- Plote um gráfico com todas as funções abaixo:

a) $f(n) = n^3$

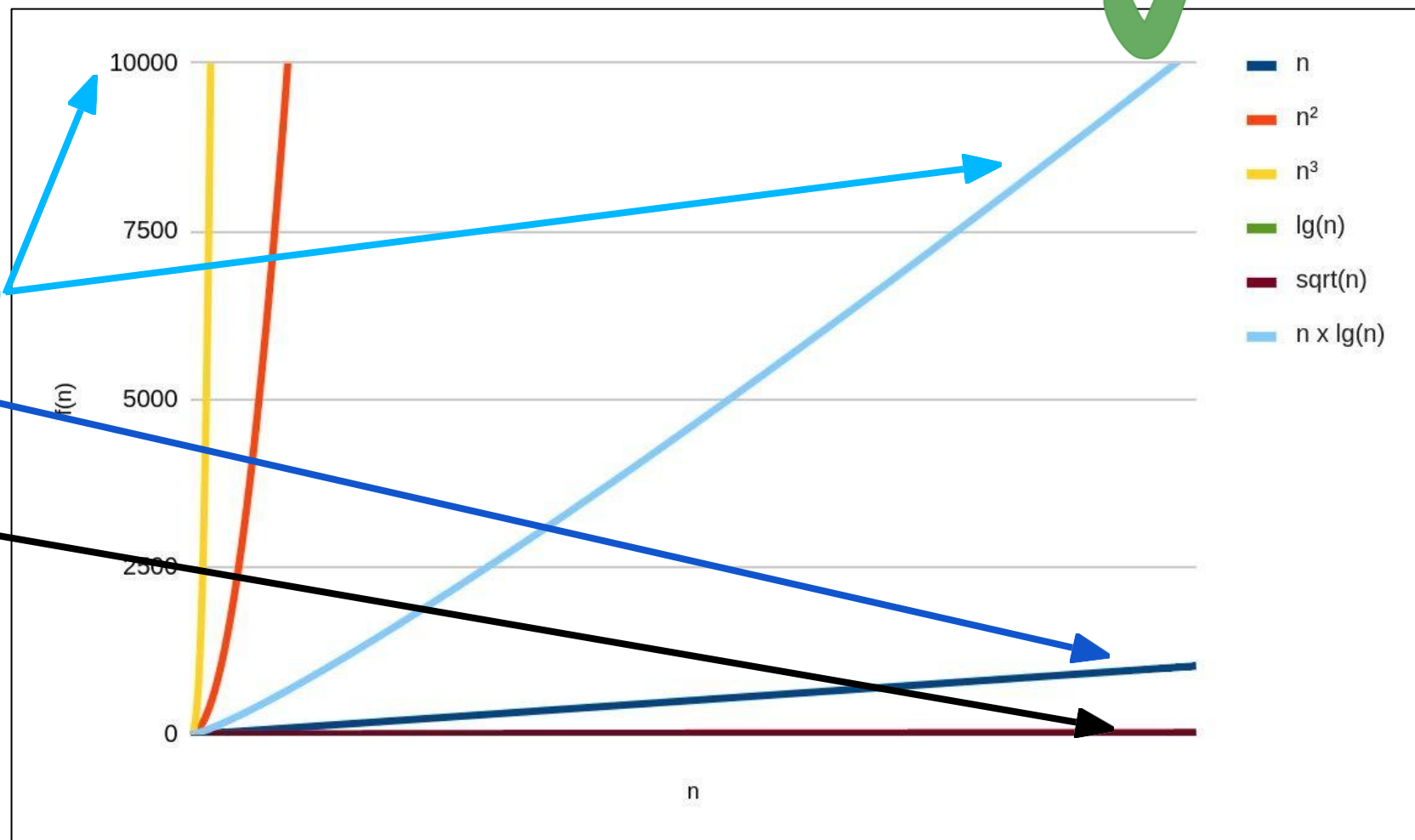
b) $f(n) = n^2$

c) $f(n) = n \times \lg(n)$

d) $f(n) = n$

e) $f(n) = \text{sqrt}(n)$

f) $f(n) = \lg(n)$



Exercício Resolvido (2)

- Plote um gráfico com todas as funções abaixo:

a) $f(n) = n^3$

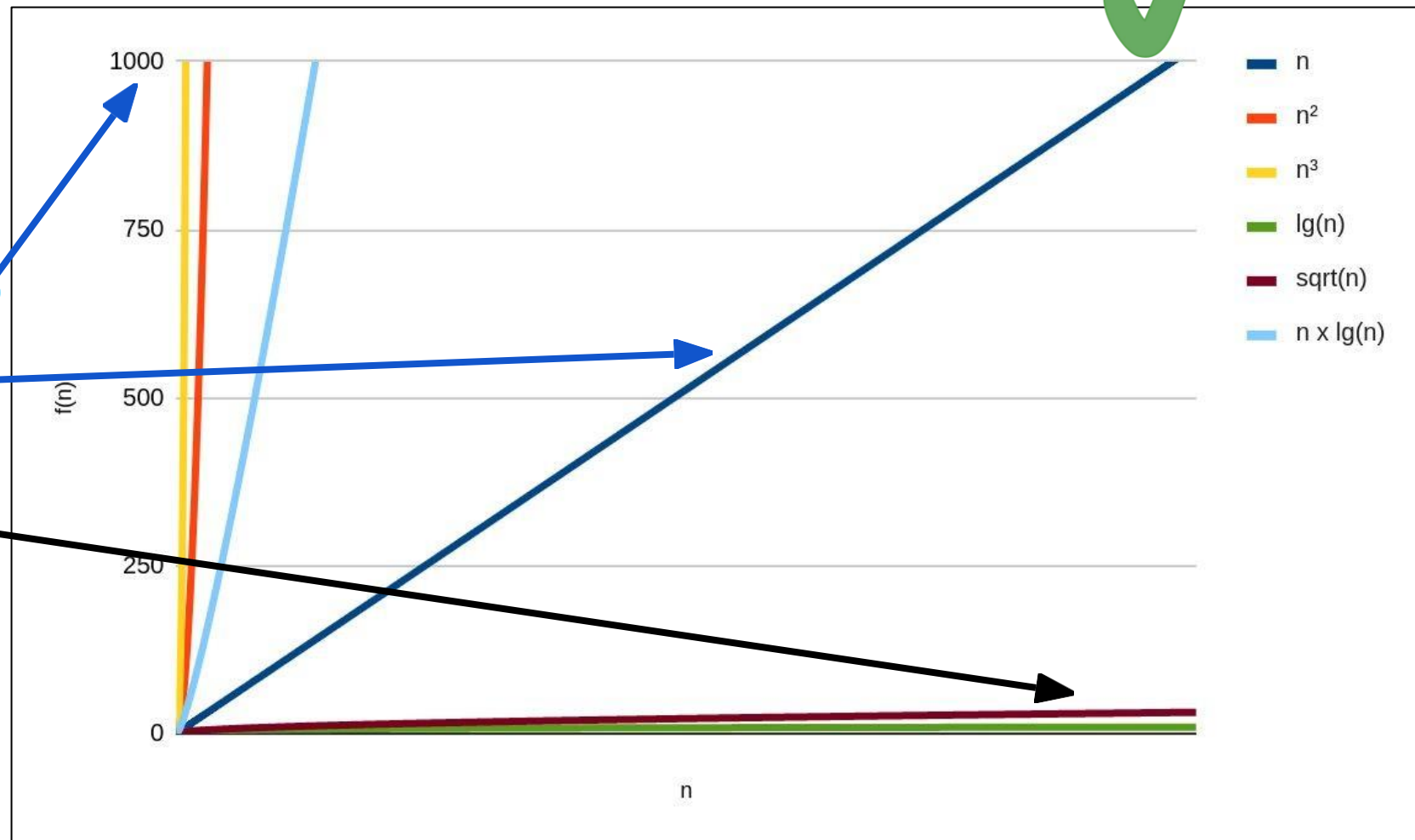
b) $f(n) = n^2$

c) $f(n) = n \times \lg(n)$

d) $f(n) = n$

e) $f(n) = \text{sqrt}(n)$

f) $f(n) = \lg(n)$



Exercício Resolvido (2)

- Plote um gráfico com todas as funções abaixo:

a) $f(n) = n^3$

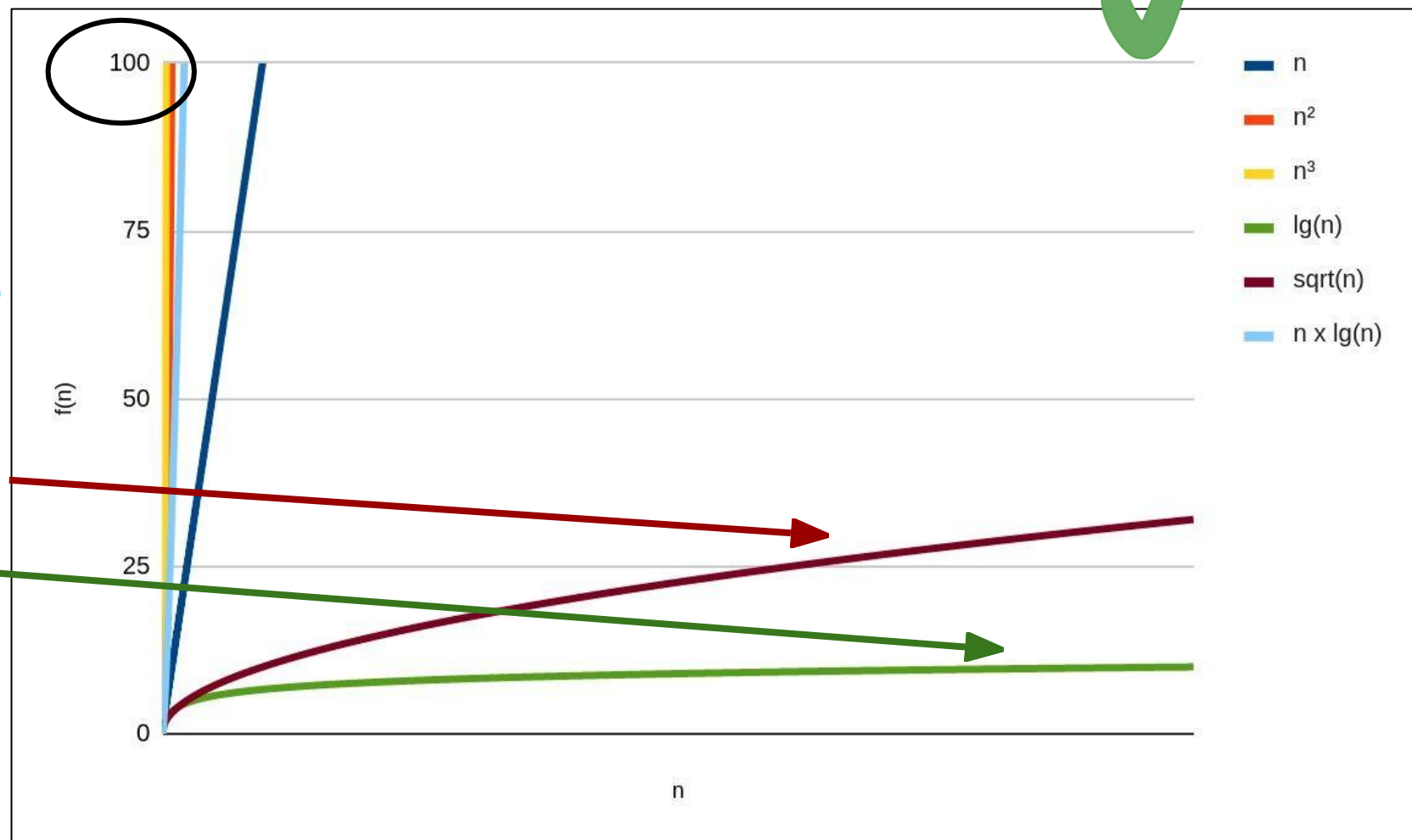
b) $f(n) = n^2$


c) $f(n) = n \times \lg(n)$

d) $f(n) = n$

e) $f(n) = \text{sqrt}(n)$

f) $f(n) = \lg(n)$



- Potência, Logaritmo, Piso e Teto, e Função
- **Contagem de operações** 
- Aspectos da análise de algoritmos
- Função de complexidade
- Notações O , Ω e Θ

Exercício Resolvido (3)

- Calcule o número de subtrações que o código abaixo realiza:

```
...  
for (int i = 0; i < n; i++){  
    if (rand() % 2 == 0){  
        a--;  
        b.-;  
    } else {  
        c.-;  
    }  
}
```

Exercício Resolvido (3)

- Calcule o número de subtrações que o código abaixo realiza:



```
...  
for (int i = 0; i < n; i++){  
    if (rand() % 2 == 0){  
        a--;  
        b.-;  
    } else {  
        c.-;  
    }  
}
```

//Melhor caso: $f(n) = n$, logo, $O(n)$, $\Omega(n)$ e $\Theta(n)$
//Pior caso: $f(n) = 2n$, logo, $O(n)$, $\Omega(n)$ e $\Theta(n)$

Cenários Possíveis

- **Melhor caso**: menor “tempo de execução” para todas entradas possíveis de tamanho n
- **Pior caso**: maior “tempo de execução” para todas entradas possíveis
- **Caso médio (ou esperado)**: média dos tempos de execução para todas as entradas possíveis (abordado em FPAA)

Contagem de Operações com Condicional

- Será o custo da condição mais ou o da lista de verdadeira ou o da falsa

```
if ( condição() ){  
    listaVerdadeiro();  
} else {  
    listaFalso();  
}
```

Melhor caso: $\text{condição()} + \text{mínimo}(\text{listaVerdadeiro()}, \text{listaFalso}())$

Pior caso: $\text{condição()} + \text{máximo}(\text{listaVerdadeiro()}, \text{listaFalso}())$

Exercício Resolvido (4)

- Calcule o número de subtrações que o código abaixo realiza:

```
...  
for (int i = 3; i < n; i++){  
    a--;  
}
```

Exercício Resolvido (4)

- Calcule o número de subtrações que o código abaixo realiza:



```
...  
for (int i = 3; i < n; i++){  
    a--;  
}  
//n - 3 subtrações  
// Logo,  $O(n)$ ,  $\Omega(n)$  e  $\Theta(n)$ 
```

Se $n = 6$, temos subtrações quando i vale 3, 4, 5 ($6 - 3 = 3$, vezes)

$n = 7$

3, 4, 5, 6 ($7 - 3 = 4$ vezes)

....

$n = 10$

3, 4, 5, 6, 7, 8, 9 ($10 - 3 = 7$ vezes)

Contagem de Operações com Repetição

- Será o custo da condição mais o número de iterações multiplicado pela soma dos custos da condição e da lista a ser repetida

```
while ( condição() ){  
    lista();  
}
```

Custo: $\text{condição()} + n * (\text{lista()} + \text{condição()})$, onde n é o número de vezes que o laço será repetido

Contagem de Operações com Repetição

- Será o número ***n*** de iterações multiplicado pela soma dos custos da lista de comandos e da condição

```
do {  
    lista();  
} while ( condição );
```

Custo: $n \times (\text{condição} + \text{lista}())$, onde n é o número de vezes que o laço será repetido

Exercício Resolvido (5)


- Calcule o número de multiplicações que o código abaixo realiza:

```
for (int i = n; i > 0; i /= 2)  
    a *= 2;
```

Exercício Resolvido (5)

- Calcule o número de multiplicações que o código abaixo realiza:

```
for (int i = n; i > 0; i /= 2)  
    a *= 2;
```



Quando n é uma potência de 2, realizamos $\lg(n) + 1$ multiplicações

Se $n = 8$, efetuamos a multiplicação quando i vale 8, 4, 2, 1

$n = 16$, 16, 8, 4, 2, 1

$n = 32$, 32, 16, 8, 4, 2, 1

Exercício Resolvido (5)

- Calcule o número de multiplicações que o código abaixo realiza:

```
for (int i = n; i > 0; i /= 2)
    a *= 2;
```

Para um valor qualquer de n ,
temos $\lfloor \lg(n) \rfloor + 1$ multiplicações,
logo, $O(\lg n)$, $\Omega(\lg n)$ e $\Theta(\lg n)$

$n = 7,$	7, 3, 1
Se $n = 8,$	efetuamos a multiplicação quando i vale	8, 4, 2, 1
$n = 9,$	9, 4, 2, 1
$n = 15,$	15, 7, 3, 1
$n = 16,$	16, 8, 4, 2, 1
$n = 17,$	17, 8, 4, 2, 1
$n = 31,$	31, 15, 7, 3, 1
$n = 32,$	32, 16, 8, 4, 2, 1
$n = 33,$	33, 16, 8, 4, 2, 1

Contagem de Operações com Repetição

- Quando tivermos uma estrutura de repetição em que o escopo de busca é sistematicamente dividido pela metade, temos um custo logarítmico

```
for (int i = n; i > 0; i /= 2){  
    lista();  
}
```

Exercício Resolvido (7)

- Encontre o menor valor em um *array* de inteiros



```
int min = array[0];  
  
for (int i = 1; i < n; i++){  
    if (min > array[i]){  
        min = array[i];  
    }  
}
```

1º) Qual é a operação relevante?

R: Comparação entre elementos do *array*

2º) Quantas vezes ela será executada?

R: Se tivermos n elementos: $T(n) = n - 1$

Exercício Resolvido (7)

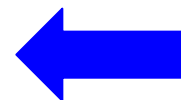
- Encontre o menor valor em um *array* de inteiros



```
int min = array[0];  
  
for (int i = 1; i < n; i++){  
    if (min > array[i]){  
        min = array[i];  
    }  
}
```

3º) O nosso $T(n) = n - 1$ é para qual dos três casos?

- Potência, Logaritmo, Piso e Teto, e Função
- Contagem de operações
- **Aspectos da análise de algoritmos**
- Função de complexidade
- Notações O , Ω e Θ



Restrição dos Algoritmos

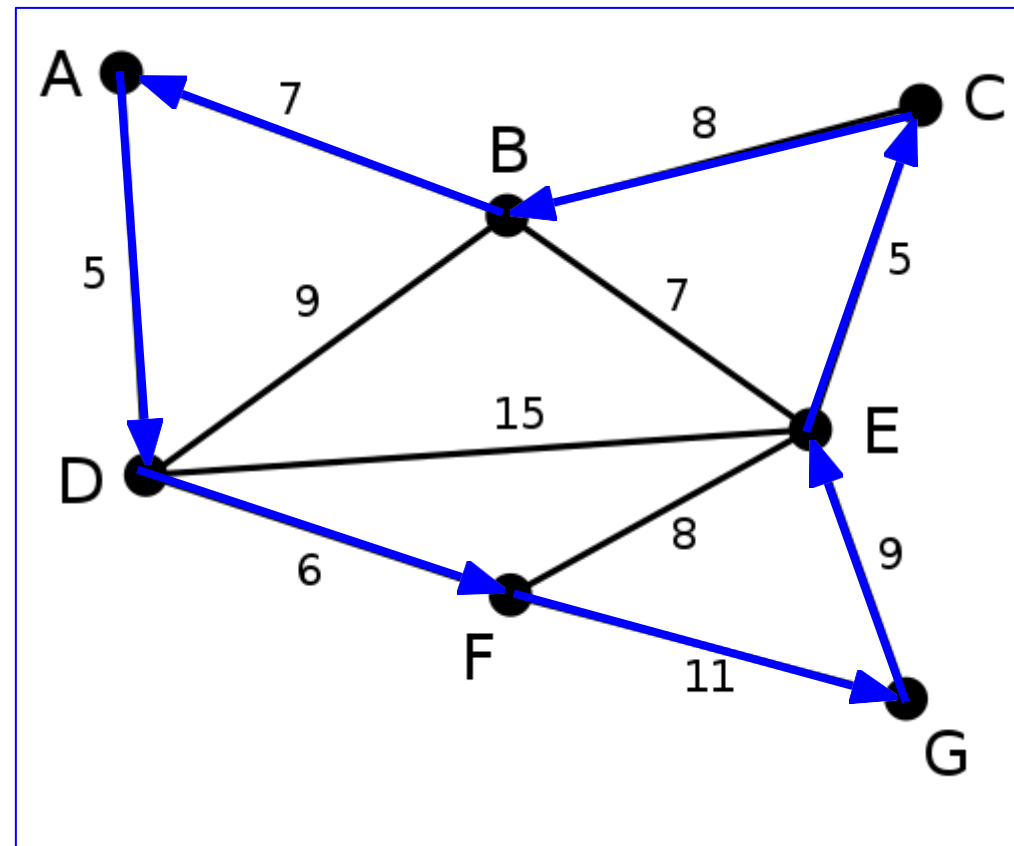
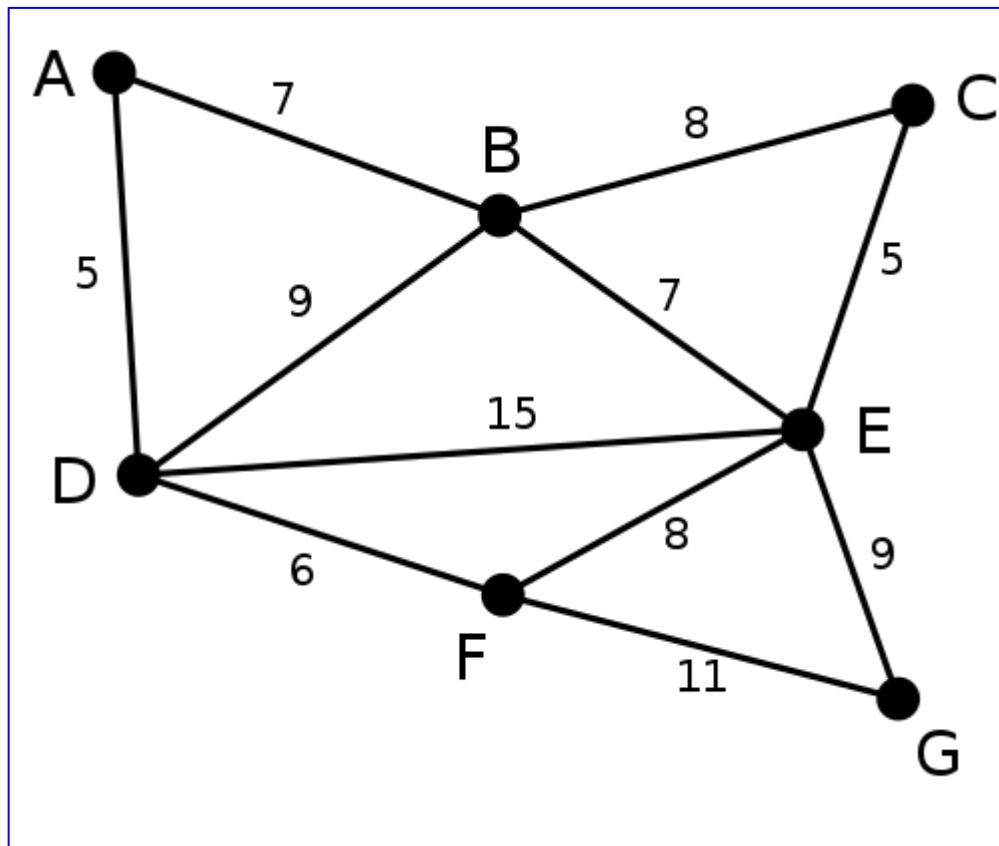
- Nossos algoritmos devem ser implementados em um computador
- Restrições do computador: capacidade computacional e armazenamento
- Logo, devemos analisar a complexidade de se implementar algoritmos

Um algoritmo que leva séculos para terminar é uma opção inadequada



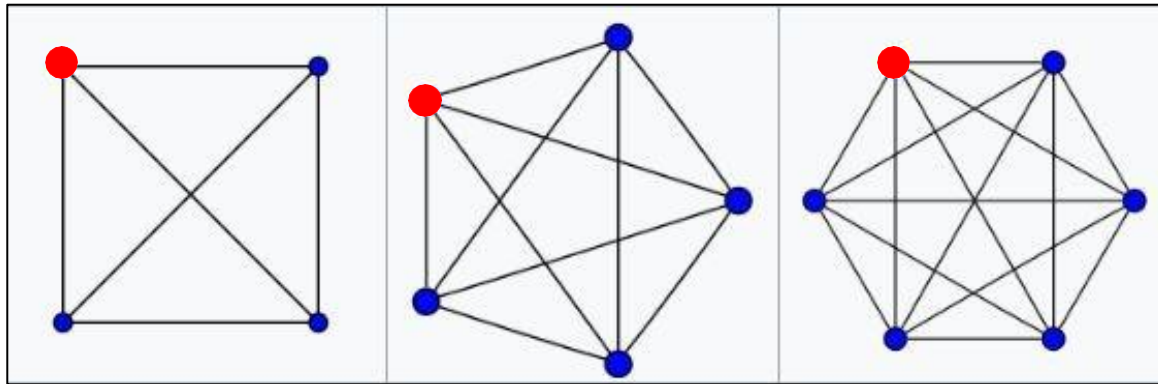
Exemplo (Rascunho) de Algoritmo NP-Completo

- Problema do Caixeiro Viajante



Exemplo (Rascunho) de Algoritmo NP-Completo

- Problema do Caixeiro Viajante



Número de combinações:

$$\overline{3} \quad \overline{2} \quad \overline{1}$$

$$\overline{4} \quad \overline{3} \quad \overline{2} \quad \overline{1}$$

$$\overline{5} \quad \overline{4} \quad \overline{3} \quad \overline{2} \quad \overline{1}$$

Exemplo (Rascunho) de Algoritmo NP-Completo

- Rascunho do algoritmo força bruta para encontrar a solução ótima do PCV

Número de cidades	Tempo de execução
5	5 s
6	$5 \times (5s) = 25 \text{ s}$
7	$6 \times (25s) = 150 \text{ s} = 2,5 \text{ min}$
8	$7 \times (2,5 \text{ min}) = 17,5 \text{ min}$
9	$8 \times (17,5 \text{ min}) = 140 \text{ min} = 2,34 \text{ h}$
10	$9 \times (2,34 \text{ h}) = 21 \text{ h}$
11	$10 \times (21 \text{ dias}) = 210 = 8,75 \text{ dias}$
12	$11 \times (8,75 \text{ dias}) = 96,25 \text{ dias}$
13	$12 \times (96,25 \text{ dias}) = 1155 = 3,15 \text{ anos}$
14	$13 \times (3,15 \text{ anos}) = 41,02 \text{ anos}$
15	$14 \times (41,02 \text{ anos}) = 5,74 \text{ séculos}$
16	$15 \times (5,74 \text{ séculos}) = 8,6 \text{ milênios}$

Exemplo (Rascunho) de Algoritmo NP-Completo

- Rascunho do algoritmo força bruta para encontrar a solução ótima do PCV

Número de cidades	Tempo de execução
Observação (1): Na verdade, a solução ótima para o PCV é duas vezes mais rápida que a apresentada, contudo, isso é “indiferente” na tendência de crescimento	
9	$8 \times (17,5 \text{ min}) = 140 \text{ min} = 2,34 \text{ h}$
10	$9 \times (2,34 \text{ h}) = 21 \text{ h}$
Observação (2): Se tivermos um computador 100 vezes mais rápido, isso também será “indiferente” na tendência de crescimento	
15	$14 \times (41,02 \text{ anos}) = 5,74 \text{ séculos}$
16	$15 \times (5,74 \text{ séculos}) = 8,6 \text{ milênios}$

Métricas para a Análise de Complexidade

- Tempo de execução
- Espaço de memória ocupado
- Outros...

Tipos de Análise de Complexidade

- **Análise de um algoritmo particular**: analisamos o custo de um algoritmo específico para um problema específico
- **Análise de uma classe de algoritmos**: analisamos o menor custo possível para resolver um problema específico
 - **Limite da família de algoritmos**, nível mínimo de dificuldade para ser resolvido

Como Medir o Custo de um Algoritmo



Restrições no Modelo do Cronômetro

- Hardware
- Arquitetura
- Sistema Operacional
- Linguagem
- Compilador

Como Medir o Custo de um Algoritmo


Modelo



Matemático

Modelo Matemático para Contar Operações

- Determinamos e contamos as operações relevantes. Em AEDs, quase sempre, comparações entre registros (elementos do *array*)
- O custo total de um algoritmo é igual a soma do custo de suas operações
- Desconsideramos sobrecargas de gerenciamento de memória ou E/S
- A menos que dito o contrário, consideramos o pior caso
- Precisamos definir a função de complexidade

- Potência, Logaritmo, Piso e Teto, e Função
- Contagem de operações
- Aspectos da análise de algoritmos
- **Função de complexidade** 
- Notações O , Ω e Θ

Algumas Funções de Complexidade

- **Função de complexidade de tempo** mede o tempo (número de execuções da operação relevante) de execução do algoritmo para um problema de tamanho n
- **Função de complexidade de espaço** mede a quantidade de memória necessária para executar um algoritmo de tamanho n

Como Calcular a Complexidade de um Algoritmo



Como Calcular a Complexidade de um Algoritmo

- Da mesma forma que calculamos o custo de um churrasco:
 - Carne: 400 gramas por pessoa (preço médio do kg R\$ 20,00 - picanha, asinha, coraçãozinho ...)
 - Cerveja: 1,2 litros por pessoa (litro R\$ 3,80)
 - Refrigerante: 1 litro por pessoa (Garrafa 2 litros R\$ 3,50)

Exercício: Monte a função de complexidade (ou custo) do nosso churrasco.

Como Calcular a Complexidade de um Algoritmo

- Da mesma forma que calculamos o custo de um churrasco:
 - Carne: 400 gramas por pessoa (preço médio do kg R\$ 20,00 - picanha, asinha, coraçãozinho ...)
 - Cerveja: 1,2 litros por pessoa (litro R\$ 3,80)
 - Refrigerante: 1 litro por pessoa (Garrafa 2 litros R\$ 3,50)

Exercício: Monte a função de complexidade (ou custo) do nosso churrasco.

$$\begin{aligned}f(n) &= n * \frac{400}{1000} * 20 + n * 1,2 * 3,8 + n * 1 * \frac{3,5}{2} \\&= 14,31 * n\end{aligned}$$

Como Calcular a Complexidade de um Algoritmo

- Da mesma forma que calculamos o custo de uma viagem:
 - Passagem:
 - Hotel:
 - Saídas:

Cálculo de Complexidade para Condicional

- Será o custo da condição mais ou o da lista de verdadeira ou o da falsa

```
if ( condição() ){  
    listaVerdadeiro();  
}  
else {  
    listaFalso();  
}
```

Melhor caso: $\text{condição()} + \min(\text{listaVerdadeiro()}, \text{listaFalso}())$

Pior caso: $\text{condição()} + \max(\text{listaVerdadeiro()}, \text{listaFalso}())$

Cálculo de Complexidade para Repetição

- Será o custo da condição mais o número de interações multiplicado pela soma dos custos da condição e da lista a ser repetida

```
while ( condição() ){  
    lista();  
}
```

Custo: $\text{condição()} + n * (\text{lista()} + \text{condição()})$, onde n é o número de vezes que o laço será repetido

Cálculo de Complexidade

- Outros laços: sempre consideramos o limite superior
- Métodos: consideramos o custo do método
- Métodos recursivos: utilizamos equações de recorrência (Vocês verão em FPAA)

Algoritmo Ótimo

- Algoritmo cujo custo é igual ao menor custo possível

Exercício Resolvido (8): Encontrar Mínimo

```
int min = array[0];  
  
for (int i = 1; i < n; i++){  
    if (min > array[i]){  
        min = array[i];  
    }  
}
```

1º) Qual é a operação relevante?

R: Comparação entre elementos do *array*

2º) Quantas vezes ela será executada?

R: Se tivermos n elementos: $T(n) = n - 1$

3º) O nosso $T(n) = n - 1$ é para qual dos três casos?

R: Para os três casos

4º) O nosso algoritmo é ótimo? Por que?

Exercício Resolvido (8): Encontrar Mínimo

```
int min = array[0];  
  
for (int i = 1; i < n; i++){  
    if (min > array[i]){  
        min = array[i];  
    }  
}
```

1º) Qual é a operação relevante?



R: Comparação entre elementos do *array*

2º) Quantas vezes ela será executada?

R: Se tivermos n elementos: $T(n) = n - 1$

3º) O nosso $T(n) = n - 1$ é para qual dos três casos?

R: Para os três casos

4º) O nosso algoritmo é ótimo? Por que?

R: Sim porque temos que testar todos os elementos para garantir nossa resposta

Exercício Resolvido (9): Pesquisa Sequencial

```
boolean resp = false;

for (int i = 0; i < n; i++){
    if (array[i] == x){
        resp = true;
        i = n;
    }
}
```

1º) Qual é a operação relevante?

R: Comparação entre elementos do array

2º) Quantas vezes ela será executada?

R: Melhor caso: $f(n) = 1$

Pior caso: $f(n) = n$

Caso médio: $f(n) = (n + 1) / 2$

3º) O nosso algoritmo é ótimo? Por que?

Exercício Resolvido (9): Pesquisa Sequencial

```
boolean resp = false;

for (int i = 0; i < n; i++){
    if (array[i] == x){
        resp = true;
        i = n;
    }
}
```

1º) Qual é a operação relevante?

R: Comparação entre elementos do array

2º) Quantas vezes ela será executada?

R: Melhor caso: $f(n) = 1$

Pior caso: $f(n) = n$

Caso médio: $f(n) = (n + 1) / 2$

3º) O nosso algoritmo é ótimo? Por que?

R: Sim porque temos que testar todos os elementos para garantir nossa resposta



Exercício (1)

- Encontre o maior e menor valores em um *array* de inteiros e, em seguida, encontre a função de complexidade de tempo para sua solução

Exercício (2)

- Problema: encontrar o valores mínimo e máximo em um vetor

```
public static void minmax1(int []vet, out int min, out int max){  
    int i;  
    min = vet[0];  
    max = vet[0];  
    for(i=1; i<vet.Length; i++){  
        if(vet[i] < min){  
            min = vet[i];  
        }  
        if(vet[i] > max){  
            max = vet[i];  
        }  
    }  
}
```

melhor caso: $f(n) = 2(n-1)$

pior caso: $f(n) = 2(n-1)$

caso médio: $f(n) = 2(n-1)$

Exercício (2)

- Se $\text{vet}[i] < \text{min}$, então não precisamos checar se $\text{vet}[i] > \text{max}$

```
public static void minmax2(int []vet, out int min, out int max){  
    int i;  
    min = vet[0];  
    max = vet[0];  
    for(i=1; i<vet.Length; i++){  
        if(vet[i] < min){  
            min = vet[i];  
        }  
        else if(vet[i] > max){  
            max = vet[i];  
        }  
    }  
}
```

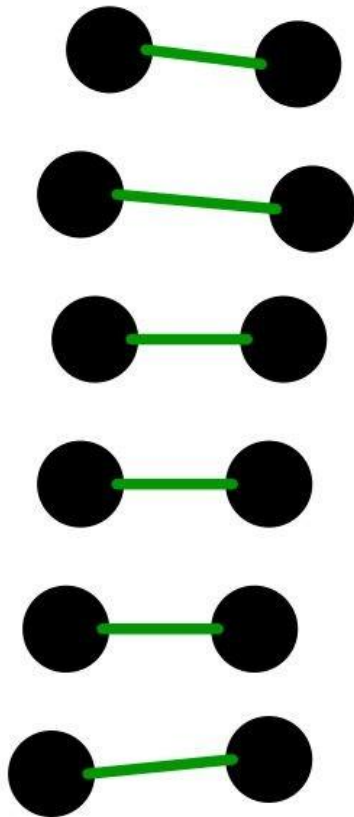
melhor caso: (decrescente) $f(n) = n-1$

pior caso: (crescente) $f(n) = 2(n-1)$

caso médio: (aleatório) $f(n) > 3(n-1)/2$

Exercício (2)

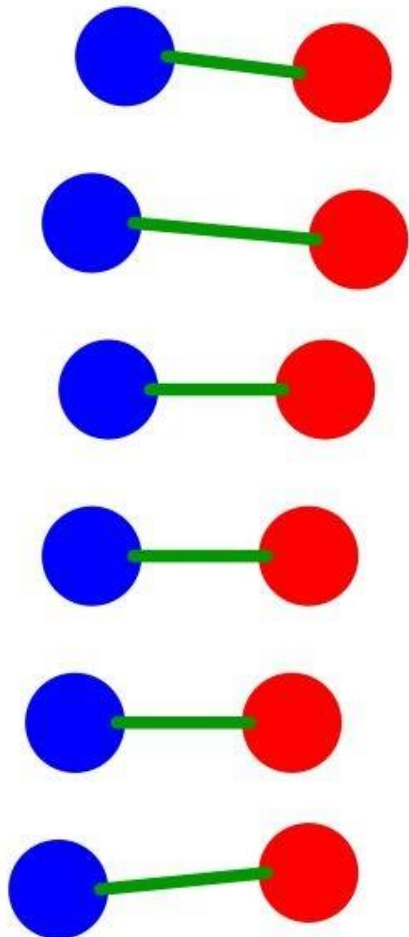
- Dá pra melhorar?



Comparar elementos par-a-par
Custo: $n/2$ comparações

Exercício (2)

• Dá pra melhorar?



- Comparar elementos par-a-par
 - Custo: $n/2$ comparações
- Elementos vermelhos são maiores que os azuis
- Encontrar o máximo entre os elementos vermelhos
 - Custo: $n/2$ comparações
- Encontrar o mínimo entre os elementos azuis
 - Custo: $n/2$ comparações

Exercício (2)

• Algoritmo Ótimo

```
using System;

class Program{

    public static void minmax3(int []vet, out int min, out int max){
        int i, a, v;
        min = Int32.MaxValue;
        max = Int32.MinValue;
        for(i=0; i<vet.Length-1; i+=2){
            if(vet[i] < vet[i+1]){
                a = i; v = i+1;
            }else{
                a = i+1; v = i;
            }
            if(vet[a] < min){
                min = vet[a];
            }
            if(vet[v] > max){
                max = vet[v];
            }
        }
    }
}
```

melhor caso:

$$f(n) = 3n/2$$

pior caso:

$$f(n) = 3n/2$$

caso médio:

$$f(n) = 3n/2$$

Exercício (2)

•Análise

Algoritmo	f(n)		
	Melhor caso	Pior caso	Caso médio
MinMax1	$2(n-1)$	$2(n-1)$	$2(n-1)$
MinMax2	$n-1$	$2(n-1)$	$> 3(n-1)/2$
MinMax3	$3n/2$	$3n/2$	$3n/2$

Exercício Resolvido (10)


- Um aluno deve procurar um valor em um *array* de números reais. Ele tem duas alternativas. Primeiro, executar uma pesquisa sequencial. Segundo, ordenar o *array* e, em seguida, aplicar uma pesquisa binária. O que fazer?

Exercício Resolvido (10)

- Um aluno deve procurar um valor em um *array* de números reais. Ele tem duas alternativas. Primeiro, executar uma pesquisa sequencial. Segundo, ordenar o *array* e, em seguida, aplicar uma pesquisa binária. O que fazer?

O aluno deve escolher a primeira opção, pois a pesquisa sequencial tem custo $\Theta(n)$. A segunda opção tem custo $\Theta(n \cdot \lg n)$ para ordenar mais $\Theta(\lg n)$ para a pesquisa binária



- Potência, Logaritmo, Piso e Teto, e Função
- Contagem de operações
- Aspectos da análise de algoritmos
- Função de complexidade
- **Notações O , Ω e Θ** 

Noção sobre as Notações O , Ω e Θ

- Regras gerais
- Definições
- Operações

Regras Gerais das Notações O , Ω e Θ

- Consideramos apenas a maior potência
- Ignoramos os coeficientes

Diferença entre as Notações O , Ω e Θ

- O é o limite superior
- Ω é o limite inferior
- Θ é o limite justo

Diferença entre as Notações O , Ω e Θ

- **O é o limite superior**, logo, se um algoritmo é $O(f(n))$, ele também será $O(g(n))$ para toda função $g(n)$ tal que “ $g(n)$ é **maior** que $f(n)$ ”
- **Ω é o limite inferior**, logo, se um algoritmo é $\Omega(f(n))$, ele também será $\Omega(g(n))$ para toda função $g(n)$ tal que “ $g(n)$ é **menor** que $f(n)$ ”
- **Θ é o limite justo**, logo, $g(n)$ é $O(f(n))$ *and* $\Omega(f(n))$ se e somente se $g(n)$ é $\Theta(f(n))$

Exercício Resolvido (11)

• Responda se as afirmações são verdadeiras ou falsas:

- a) $3n^2 + 5n + 1$ é $O(n)$:
- b) $3n^2 + 5n + 1$ é $O(n^2)$:
- c) $3n^2 + 5n + 1$ é $O(n^3)$:
- d) $3n^2 + 5n + 1$ é $\Omega(n)$:
- e) $3n^2 + 5n + 1$ é $\Omega(n^2)$:
- f) $3n^2 + 5n + 1$ é $\Omega(n^3)$:
- g) $3n^2 + 5n + 1$ é $\Theta(n)$:
- h) $3n^2 + 5n + 1$ é $\Theta(n^2)$:
- i) $3n^2 + 5n + 1$ é $\Theta(n^3)$:

Exercício Resolvido (11)

• Responda se as afirmações são verdadeiras ou falsas:

a) $3n^2 + 5n + 1$ é $O(n)$:

b) $3n^2 + 5n + 1$ é $O(n^2)$: verdadeira

c) $3n^2 + 5n + 1$ é $O(n^3)$:

d) $3n^2 + 5n + 1$ é $\Omega(n)$:

e) $3n^2 + 5n + 1$ é $\Omega(n^2)$: verdadeira

f) $3n^2 + 5n + 1$ é $\Omega(n^3)$:

g) $3n^2 + 5n + 1$ é $\Theta(n)$:

h) $3n^2 + 5n + 1$ é $\Theta(n^2)$: verdadeira

i) $3n^2 + 5n + 1$ é $\Theta(n^3)$:



Exercício Resolvido (11)

• Responda se as afirmações são verdadeiras ou falsas:

- a) $3n^2 + 5n + 1$ é $O(n)$:
- b) $3n^2 + 5n + 1$ é $O(n^2)$: verdadeira
- c) $3n^2 + 5n + 1$ é $O(n^3)$: verdadeira
- d) $3n^2 + 5n + 1$ é $\Omega(n)$: verdadeira
- e) $3n^2 + 5n + 1$ é $\Omega(n^2)$: verdadeira
- f) $3n^2 + 5n + 1$ é $\Omega(n^3)$:
- g) $3n^2 + 5n + 1$ é $\Theta(n)$:
- h) $3n^2 + 5n + 1$ é $\Theta(n^2)$: verdadeira
- i) $3n^2 + 5n + 1$ é $\Theta(n^3)$:



Exercício Resolvido (11)

• Responda se as afirmações são verdadeiras ou falsas:

- a) $3n^2 + 5n + 1$ é $O(n)$: falsa
- b) $3n^2 + 5n + 1$ é $O(n^2)$: verdadeira
- c) $3n^2 + 5n + 1$ é $O(n^3)$: verdadeira
- d) $3n^2 + 5n + 1$ é $\Omega(n)$: verdadeira
- e) $3n^2 + 5n + 1$ é $\Omega(n^2)$: verdadeira
- f) $3n^2 + 5n + 1$ é $\Omega(n^3)$: falsa
- g) $3n^2 + 5n + 1$ é $\Theta(n)$:
- h) $3n^2 + 5n + 1$ é $\Theta(n^2)$: verdadeira
- i) $3n^2 + 5n + 1$ é $\Theta(n^3)$:



Exercício Resolvido (11)

• Responda se as afirmações são verdadeiras ou falsas:

- a) $3n^2 + 5n + 1$ é $O(n)$: falsa
- b) $3n^2 + 5n + 1$ é $O(n^2)$: verdadeira
- c) $3n^2 + 5n + 1$ é $O(n^3)$: verdadeira
- d) $3n^2 + 5n + 1$ é $\Omega(n)$: verdadeira
- e) $3n^2 + 5n + 1$ é $\Omega(n^2)$: verdadeira
- f) $3n^2 + 5n + 1$ é $\Omega(n^3)$: falsa
- g) **$3n^2 + 5n + 1$ é $\Theta(n)$: falsa**
- h) $3n^2 + 5n + 1$ é $\Theta(n^2)$: verdadeira
- i) **$3n^2 + 5n + 1$ é $\Theta(n^3)$: falsa**



Exercício (3)

- Preencha verdadeiro ou falso na tabela abaixo:

	$O(1)$	$O(\lg n)$	$O(n)$	$O(n \cdot \lg(n))$	$O(n^2)$	$O(n^3)$	$O(n^5)$	$O(n^{20})$
$f(n) = \lg(n)$								
$f(n) = n \cdot \lg(n)$								
$f(n) = 5n + 1$								
$f(n) = 7n^5 - 3n^2$								
$f(n) = 99n^3 - 1000n^2$								
$f(n) = n^5 - 99999n^4$								

Exercício (4)

- Preencha verdadeiro ou falso na tabela abaixo:

	$\Omega(1)$	$\Omega(\lg n)$	$\Omega(n)$	$\Omega(n \cdot \lg(n))$	$\Omega(n^2)$	$\Omega(n^3)$	$\Omega(n^5)$	$\Omega(n^{20})$
$f(n) = \lg(n)$								
$f(n) = n \cdot \lg(n)$								
$f(n) = 5n + 1$								
$f(n) = 7n^5 - 3n^2$								
$f(n) = 99n^3 - 1000n^2$								
$f(n) = n^5 - 99999n^4$								

Exercício (5)

- Preencha verdadeiro ou falso na tabela abaixo:

	$\Theta(1)$	$\Theta(\lg n)$	$\Theta(n)$	$\Theta(n \cdot \lg(n))$	$\Theta(n^2)$	$\Theta(n^3)$	$\Theta(n^5)$	$\Theta(n^{20})$
$f(n) = \lg(n)$								
$f(n) = n \cdot \lg(n)$								
$f(n) = 5n + 1$								
$f(n) = 7n^5 - 3n^2$								
$f(n) = 99n^3 - 1000n^2$								
$f(n) = n^5 - 99999n^4$								

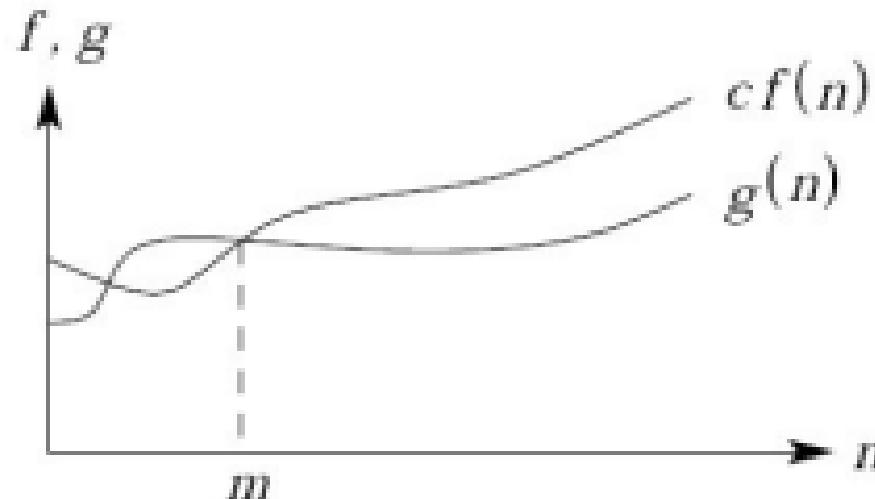
Notação O

- A notação O é utilizada para descrever a tendência de uma função
- Ela é muito utilizada em computação para classificar algoritmos de acordo com a taxa de crescimento de suas funções

Dominação Assintótica

Uma função $f(n)$ *domina* assintoticamente outra função $g(n)$ se existem duas constantes positivas c e m tais que, para $n \geq m$, temos $|g(n)| \leq c \times |f(n)|$

Ou seja, para números grandes $cf(n)$ será sempre maior que $g(n)$, para alguma constante c



Notação O para dominação Assintótica

Quando uma função $g(n) = O(f(n))$, dizemos que:

$g(n)$ é O de $f(n)$ \rightarrow $g(n)$ é da ordem de no máximo $f(n)$

$f(n)$ domina $g(n)$ assintoticamente

Ex: Quando dizemos que o tempo de execução $T(n)$ de um programa é $O(n^2)$, significa que existem constantes **c** e **m** tais que, para valores de **n** \geq **m**, **$T(n) \leq cn^2$** .

A notação é importante para comparar algoritmos pois não estamos interessados em funções exatas de custo mas sim, do comportamento da função

Exemplo 1

Dadas as funções:

$$g(n) = (n+1)^2$$

$$f(n) = n^2$$

As duas funções se dominam assintoticamente pois:

$$g(n) = O(f(n)) \text{ pois } g(n) \leq 4f(n) \text{ para } n > 1$$

$$f(n) = O(g(n)) \text{ pois } f(n) \leq g(n)$$

Exemplo 2

Dadas as funções:

$$g(n) = (n+1)^2$$

$$f(n) = n^3$$

A segunda função domina a primeira

$g(n) = O(f(n))$ pois $g(n) \leq cf(n)$ para um c coerente e a partir de $n > 1$

Porém a primeira não domina a segunda:

Pois $f(n) \leq cg(n)$ seria uma afirmação falsa para qualquer c e a partir de qualquer n

Limites Fortes

$$g(n) = 3n^3 + 2n^2 + n \text{ é } O(n^3)$$

Pois $g(n) \leq 6n^3$ para $n \geq 0$

É claro que também podemos dizer que $g(n) = O(n)$, porém esta seria uma afirmação fraca;

Para analisar comportamentos de algoritmos, estamos interessados em afirmações fortes

Operações com as Notações O , Ω e Θ

$$1) f(n) = O(f(n))$$

$$2) c \times O(f(n)) = O(f(n))$$

$$3) O(f(n)) + O(f(n)) = O(f(n))$$

$$4) O(O(f(n))) = O(f(n))$$

$$5) O(f(n)) + O(g(n)) = O(\text{máximo}(f(n), g(n)))$$

$$6) O(f(n)) \times O(g(n)) = O(f(n) \times g(n))$$

$$7) f(n) \times O(g(n)) = O(f(n) \times g(n))$$

***) As mesmas propriedades são aplicadas para Ω e Θ**

Operações com as Notações O , Ω e Θ

$$f(n) = O(f(n))$$

- Uma função sempre se domina pois basta a multiplicar por uma constante para que seja maior que ela mesmo

***) As mesmas propriedades são aplicadas para Ω e Θ**

Operações com as Notações O , Ω e Θ

$$c \times O(f(n)) = O(f(n))$$

- O mesmo vale para esta função $O(f(n))$ multiplicada por uma constante, pois basta multiplicarmos $f(n)$ por uma constante ainda maior para que seja dominada

***) As mesmas propriedades são aplicadas para Ω e Θ**

Operações com as Notações O , Ω e Θ

$$O(f(n)) + O(f(n)) = O(f(n))$$

- A soma de duas funções dominadas por $f(n)$ é ainda dominada por $f(n)$ pois esta diferença pode ainda ser compensada por uma constante

***) As mesmas propriedades são aplicadas para Ω e Θ**

Operações com as Notações O , Ω e Θ

$$O(O(f(n))) = O(f(n))$$

- Se uma função é dominada por uma função dominada por $f(n)$, a primeira função é também dominada por $f(n)$

***) As mesmas propriedades são aplicadas para Ω e Θ**

Operações com as Notações O , Ω e Θ

$$O(f(n)) + O(g(n)) = O(\text{máximo}(f(n), g(n)))$$

- A soma de duas funções será dominada pela maior função que as domina

***) As mesmas propriedades são aplicadas para Ω e Θ**

Operações com as Notações O , Ω e Θ

$$O(f(n)) \times O(g(n)) = O(f(n) \times g(n))$$

$$f(n) \times O(g(n)) = O(f(n) \times g(n))$$

- A multiplicação de duas funções será dominada pela multiplicação das funções que as dominavam

***) As mesmas propriedades são aplicadas para Ω e Θ**

Exercício Resolvido (12)

Sabendo que o Algoritmo de Seleção faz $\Theta(n^2)$ comparações entre registros, quantas dessas comparações temos no código abaixo? Justifique

```
for (int i = 0; i < n; i++){  
    seleção();  
}
```

Exercício Resolvido (12)

Sabendo que o Algoritmo de Seleção faz $\Theta(n^2)$ comparações entre registros, quantas dessas comparações temos no código abaixo? Justifique



```
for (int i = 0; i < n; i++){  
    seleção();  
}
```

Neste caso, executamos o Seleção n vezes: $n \times \Theta(n^2) = \Theta(n^3)$

Exercício Resolvido (13)

Sabendo que o limite inferior da ordenação é $\Theta(n \lg n)$ e que o custo da pesquisa binária é $\Theta(\lg n)$, qual é a ordem de complexidade de uma solução em que ordenamos um *array* e efetuamos uma pesquisa binária. Justifique sua resposta

Exercício Resolvido (13)

Sabendo que o limite inferior da ordenação é $\Theta(n \lg n)$ e que o custo da pesquisa binária é $\Theta(\lg n)$, qual é a ordem de complexidade de uma solução em que ordenamos um *array* e efetuamos uma pesquisa binária. Justifique sua resposta



Neste caso, temos duas etapas e o custo total será a soma das mesmas, logo: $\Theta(n \lg n) + \Theta(\lg n) = \Theta(n \lg n)$

Classe de Algoritmos

- Constante: $O(1)$
- Logarítmico: $O(\lg n)$
- Linear: $O(n)$
- Linear-logarítmico: $O(n \lg n)$
- Quadrático: $O(n^2)$
- Cúbico: $O(n^3)$
- Exponencial: $O(c^n)$
- Fatorial: $O(n!)$

Algoritmos Polinomiais

- Um algoritmo é polinomial se é $O(n^p)$ para algum inteiro p
- Problemas com algoritmos polinomiais são considerados tratáveis
- Problemas para os quais não há algoritmos polinomiais são considerados intratáveis

Exercício Resolvido (19)

- Apresente a função e a complexidade para o número de subtrações para o pior e melhor caso

```
for (i = 0; i < n; i++) {  
    for (j = 1; j <= n; j *= 2) {  
        b--;  
    }  
}
```

Exercício Resolvido (19)

- Apresente a função e a complexidade para o número de subtrações para o pior e melhor caso

```
for (i = 0; i < n; i++) {
  for (j = 1; j <= n; j *= 2) {
    ...
  }
}
```



função

complexidade

TODOS

$$f(n) = (\lg(n) + 1) * n = n * \lg(n) + n$$

$$O(n \times \lg(n)), \Omega(n \times \lg(n)) \text{ e } \Theta(n \times \lg(n))$$

Exercício Resolvido (21)

- Classifique as funções $f_1(n) = n^2$, $f_2(n) = n$, $f_3(n) = 2^n$, $f_4(n) = (3/2)^n$, $f_5(n) = n^3$ e $f_6(n) = 1$ de acordo com o crescimento, do mais lento para o mais rápido (Khan Academy, adaptado)

Exercício Resolvido (21)

- Classifique as funções $f_1(n) = n^2$, $f_2(n) = n$, $f_3(n) = 2^n$, $f_4(n) = (3/2)^n$, $f_5(n) = n^3$ e $f_6(n) = 1$ de acordo com o crescimento, do mais lento para o mais rápido (Khan Academy, adaptado)

$$f_6(n) = 1$$

$$f_2(n) = n$$

$$f_1(n) = n^2$$

$$f_5(n) = n^3$$

$$f_4(n) = (3/2)^n$$

$$f_3(n) = 2^n$$



Exercício Resolvido (23)

- Faça a correspondência entre cada função $f(n)$ com sua $g(n)$ equivalente, em termos de Θ . Essa correspondência acontece quando $f(n) = \Theta(g(n))$ (Khan Academy, adaptado)

$f(n)$	$g(n)$
$n + 30$	n^4
$n^2 + 2n - 10$	$3n - 1$
$n^3 \cdot 3n$	$\lg(2n)$
$\lg(n)$	$n^2 + 3n$

Exercício Resolvido (23)

- Faça a correspondência entre cada função $f(n)$ com sua $g(n)$ equivalente, em termos de Θ . Essa correspondência acontece quando $f(n) = \Theta(g(n))$
(Khan Academy, adaptado)

$f(n)$	$g(n)$
$n + 30$	n^4
$n^2 + 2n - 10$	$3n - 1$
$n^3 \cdot 3n$	$\lg(2n)$
$\lg(n)$	$n^2 + 3n$